Vorlesungsmitschrift

Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

Inhaltsverzeichnis

1 Einführung (Vorlesung 1 am 17.04.)

1.1 Was sind Programmiersprachen?

Definition 1.1 (Programmiersprachen).
Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.

Memo technischer Begriff -> z.b. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

Syntax einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)
- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

Semantik (diese Vorlesung)
- Bedeutung
- Interpretation
  Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

Fokus: Formalisierung

Pragmatik Fragen nach dem Gebrauch und Zweck (Useability).
Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)

1.2 Mehrdeutigkeit in natürlichen Sprachen

Synonyme *Schloss, Schimmel, ...*
Auflösung durch Kontext (meist leicht und unproblematisch)

Satzebene *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*
Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

Entwicklung der Formalisierungsansätze

operationale Semantik *(Landin 1964)*: Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnliche wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

denotationelle Semantik *(McCarthy 1962)*: Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

axiomatische Semantik *(Hoare 1969)*: Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel: $\{Q\}P\{R\}$

1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)
Elementare Einheiten

```
// ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
Z::= 0 |1|...|MAX|-1|-2|...|MIN
// Wahrheitswerte BOOL
W::= TRUE | FALSE
// Konstanten KON
K::= Z | W
// Bezeichner bzw. Variablen mit Indizes
I::= a|b|...|z|a1|a2|...|zi
// Operatoren
OP::= + | - | * | / | mod
// boolesche Operatoren
BOP::= < | > | = | !> | !< | !=
```

Zusätzliche Einheiten *(induktiv)*

```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | READ, für T1,T2 in TERM
// boolesche Terme BT
B::= W | T1 BOP T2 | read | not B
// Befehle (Zustandstransformation) COM
C::= skip | I := T | C1; C | if B then C1 else C2 | while B do C |
    output T | output B
```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!
Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

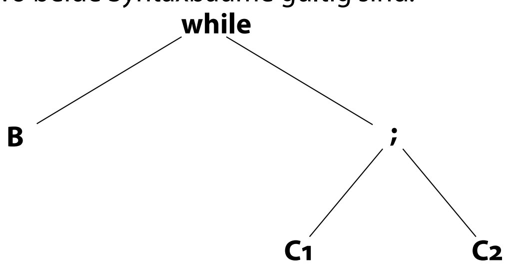
- Typkonflikte
- Fehlerbehandlung
- Rekursion

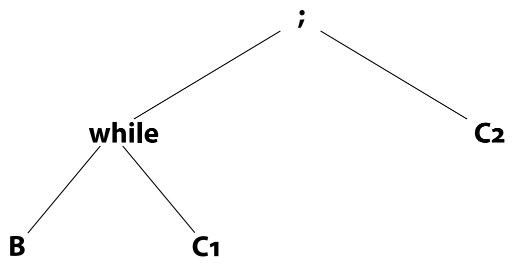
WHILE ist mehrdeutig?

Ja, das zeigt folgendes Beispiel:

```
while B do C1; C2
```

wo beide Syntaxbäume gültig sind.



und


Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

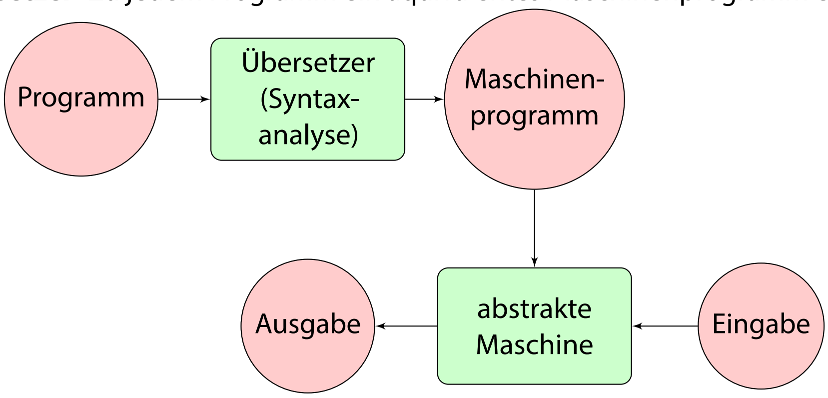
```
while B do
  C₁;
  C₂
```

2 Operationelle Semantik (Vorlesung 2 am 24.04.)

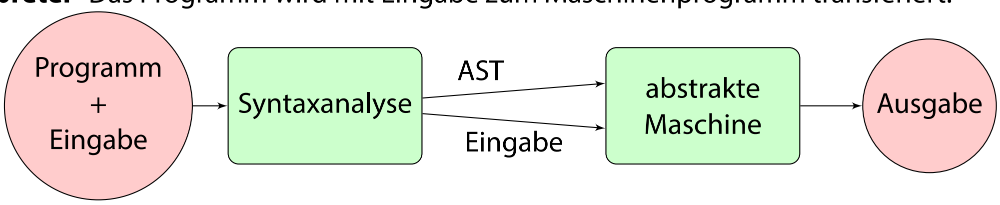
2.1 Operationelle Semantik am Beispiel der Terme

(Inhalt ist nicht im Lehrbuch!)

Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen!
Grundsätzlich gibt es zwei Methoden:
Übersetzer Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



Interpreter Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



AST: abstract syntax tree

2.2 Terme der Sprache WHILE

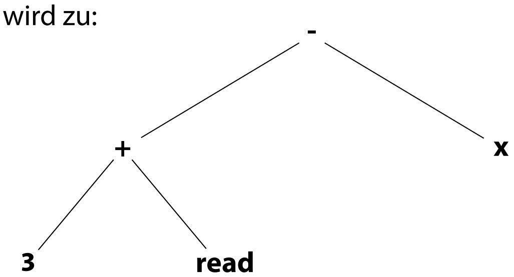
```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | READ, für T1,T2 in TERM
```

Beispiel: AST

Der Ausdruck

```
3 + read - x
```

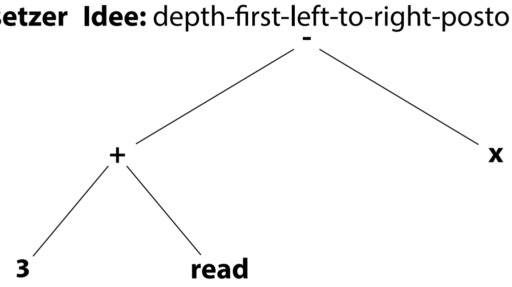
wird zu:



2.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.
Annahme: wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

Übersetzer **Idee:** depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

```
PUSH 3
READ
ADD
LOAD x
SUB
```

Zustandsveränderungen:
aktueller Zustand (siehe Bild Architektur!):

$$< \epsilon|S|8.5.... > \xrightarrow{\texttt{PUSH 3}} \quad < 3.|S|8.5.... >$$

$$\xrightarrow{\texttt{READ}} \quad < -8.3 - \epsilon|S|5.... >$$

$$\xrightarrow{\texttt{ADD}} \quad < 3 + (-8).\epsilon|S|5.... >$$

$$\xrightarrow{\texttt{LOAD x}} \quad < 2. - 5.\epsilon|S|...5 >$$

$$\xrightarrow{\texttt{SUB}} \quad < -7.\epsilon|S|5... >$$

Semantik eines Terms T zu geg. Speicher S und Eingabe E ist die Spitze des Wertekellers(STACK) nach Ausführung von trans T auf $< \epsilon|S|E >$, falls diese Ausführung fehlerfrei läuft, sonst Fehler!

Interpreter (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
  * Wertekeller W ($\in ZAHL^*$)
  * Speicher S ($S \in [ID \to ZAHL]$)
  * Kontrollkeller K ($\in (AST \cup OP)^*$)
  * Eingabe E ($\in ZAHL^*$)

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum Z durch Angabe von:

(i) einem Anfangszustand $Z_{T,S,E}$ für jeden Term T, Speicher S und Eingabe E.
(ii) eine Zustandsüberführungsfunktion $\Delta : Z \to Z$ (partiell)
(iii) Erklärung der Semantik über Iteration von Δ

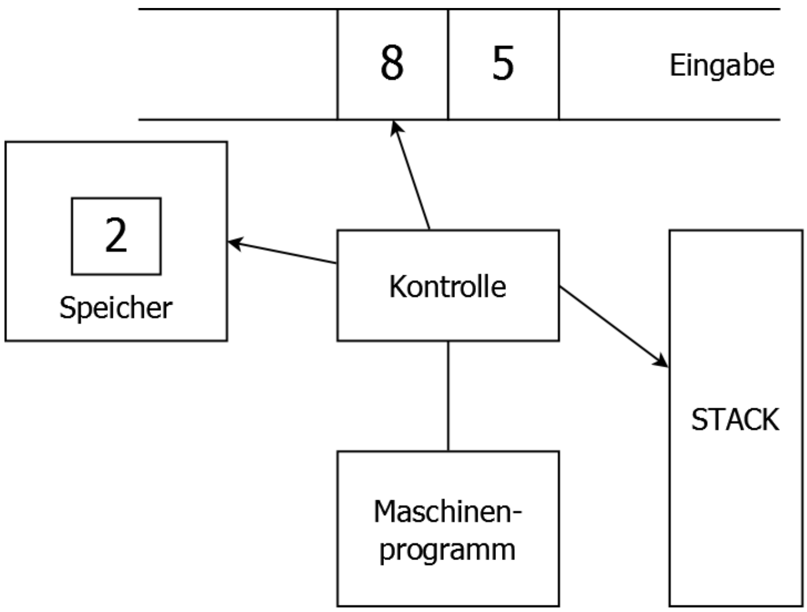
für Terme aus WHILE:

(i) $Z_{T_0,S_0,E_0} :=< \epsilon|S_0|T_0.\epsilon|E_0 >$
(ii) Δ per Induktion über die Struktur der Kontrollkellerspitze

$$
\begin{aligned}
\Delta < W|S|n.K|E > \quad &= < n.W|S|K|E > \text{ für alle } n \in ZAHL, W, S, E \text{ wie oben.}\\
\Delta < W|S|x.K|E > \quad &= < s(x).W|S|K|E > \text{ für alle } x \in ID\\
\Delta < W|S|read.K|n.E > \quad &= < n.W|S|K|E > \text{ für alle } n \in ZAHL\\
\Delta < W|S|T_1 OPT_2.K|E > \quad &= < W|S|T_1.T_2.OP.K|E >\\
\Delta < n_2.n_1.W|S|OP.K|E > \quad &= < n_1 OP n_2.W|S|K|E > n_1, n_2 \in ZAHL \text{ falls } n_1 OP n_2 \text{ definiert ist.}
\end{aligned}
$$

(iii) Die Semantik eines (beliebigen) Terms T im Bezug auf einem Speicher S und eine Eingabe E ist $n \in ZAHL$, wenn $\Delta^k Z_{T,S,E} =< n.\epsilon|S|\epsilon|E' >$ für beliebige $E' \in ZAHL^*$, undefiniert sonst!

Architektur der abst. Maschine



Befehlssatz

```
// Stack und Speicher Operationen
READ        // lese Zahl v. Eingabe, PUSH Zahl, rücke Zeiger um eins weiter
LOAD x      // PUSH Inhalt aus Speicher mit symbl. Adr. x
PUSH n      // für jede Zahl aus N lege n auf Stack
(STORE x)   // belegt Speicher mit der symbolischer Adresse x
(GOTO n)    // bedingter Sprung

// arithmetische Operationen
ADD
MULT
SUB
```

3 Operationelle Semantik (Vorlesung 3 am 08.05.)

(Mitschrift von HvB, da Autor im Urlaub. Layout modifiziert von TH.)

3.1 Allgemeines

Methodik einer Formalisierung (Interpreter) der Semantik einer Programmiersprache $\mathcal{P}$
Zur operationellen Semantik gehören insbesondere 3 Angaben:

1. Definition des Zustandsraums einer abstrakten Maschine - möglichst einfach: $\mathcal{Z}$
2. Definition einer (partiellen) Zustandsüberführungsfunktion $\Delta : \mathcal{Z} \to \mathcal{Z}$
3. Definition eines Anfangszustands $\mathcal{Z}_{P,E}$ zu jedem Programm P und Eingabe E

Aus 1.-3. ergibt sich die operationelle Semantik $\mathcal{O}$ von $\mathcal{P}$ wie folgt:
$\mathcal{O} : \mathcal{P} \to [\mathcal{E} \to \mathcal{A} \cup \{Fehler\}]$

$$\mathcal{O}(P)(E) = \begin{cases} A, & \text{falls } \exists k \in \mathbb{N} \text{ mit } \Delta^k(Z_{P,E} = \Delta^{k+1}(Z_{P,E}) \text{ und} \\ & A \text{ die Ausgabekomponente von } Z \text{ ist} \\ \text{Fehler}, & \text{falls es ein } k \in \mathbb{N} \text{ gibt mit } \Delta^k(Z_{P,E}) \text{ nicht definiert} \\ \text{undefiniert}, & \text{sonst.} \end{cases}$$

3.2 Operationelle Semantik von WHILE

1. **Der Zustandsraum $\mathcal{Z}$ ist das kartesische Produkt** $\mathcal{W} \times \mathcal{S} \times \mathcal{K} \times \mathcal{E} \times \mathcal{A}$ mit:

   Wertekeller $W \in \mathcal{W}$ ist eine Folge von Konstanten, d.h.
   $\mathcal{W} = KON^*$
   Speicher $S \in \mathcal{S}$ ist eine Funktion von Bezeichnern nach $ZAHL \cup \{frei\}$, d.h.
   $\mathcal{S} = [ID \to ZAHL]$
   Kontrollkeller $K \in \mathcal{K}$ ist ein Folge von ASTs bzw. Kontrollsymbolen, d.h.
   $\mathcal{K} = \big(TERM \cup BT \cup COM \cup OP \cup BOP \cup \{\underline{if}, \underline{while}, \underline{assign}\}\big)^*$
   Ein- und Ausgabe $E \in \mathcal{E}$ bzw. $A \in \mathcal{A}$ ist jeweils eine Folge von Konstanten, d.h.
   $\mathcal{E} = KON^*$ bzw. $\mathcal{A} = KON^*$

2. **Induktion über den Aufbau der Kontrollkellerspitze**

   a. TERM - $T := Z|I|T_1\ \underline{OP}\ T_2|\underline{read}$
   $\Delta\langle W|S|n.K|E|A\rangle = \langle n.W|S|K|E|A\rangle$ für alle $n \in ZAHL$
   $\Delta\langle W|S|x.K|E|A\rangle = \langle S(x).W|S|K|E|A\langle$ für alle $x \in ID$ mit $S(x) \neq frei$
   $\Delta\langle W|S|\underline{read}.K|n.E|A\rangle = \langle n.W|S|K|E|A\langle$ für alle $n \in ZAHL$
   $\Delta\langle W|S|T_1\underline{OP}\,T_2.K|E|A\rangle = \langle W|S|T_1.T_2.\underline{OP}.K|E|A\rangle$
   $\Delta\langle n_2.n_1.W|S|\underline{OP}.K|E|A\rangle = \langle(n_1\ \underline{OP}\ n_2).W|S|K|E|A\rangle$, falls $\underline{n_1\ OP\ n_2}$ nicht aus dem darstellbaren Zahlenbereich herausführt

   b. Boolsche Terme (ähnlich)

   c. COM - $C := \underline{skip}|I := T|C_1; C_2|\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2|\underline{while}\ B\ \underline{do}\ C|\underline{output}\ T|\underline{output}\ B$
   $\Delta\langle W|S|skip.K|E|A\rangle = \langle W|S|K|E|A\rangle$
   $\Delta\langle W|S|I := T.K|E|A\rangle = \langle W|S|T.assign.I.K|E|A\rangle$
   $\Delta\langle n.W|S|assign.I.K|E|A\rangle = \langle W|S[n/I]|K|E|A\rangle$, wobei $n \in ZAHL$ und
   $S[n/I](x) = \begin{cases} n, & \text{falls } I = x \\ S(x) & sonst \end{cases}$
   $\Delta\langle W|S|C_1; C_2.K|E|A\rangle = \langle W|S|C_1.C_2.K|E|A\rangle$ $\Delta\langle W|S|\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2.K|E|A\rangle = \langle W|S|B.\underline{if}.C_1.C_2.K|E|A\rangle$

$$\Delta\langle \underline{true}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle = \langle W|S|C_1.K|E|A|\rangle$$
$$\Delta\langle \underline{false}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle = \langle W|S|C_2.K|E|A|\rangle$$
$$\Delta\langle W|S|\underline{while}\ B\ \underline{do}\ C.K|E|A\rangle = \langle W|S|B.\underline{while}.B.C.K|E|A\rangle$$
$$\Delta\langle \underline{true}.W|S|\underline{while}.B.C.K|E|A\rangle = \langle W|S|C.B.\underline{while}.B.C.K|E|A\rangle$$
$$\Delta\langle \underline{false}.W|S|\underline{while}.B.C.K|E|A\rangle = \langle W|S|K|E|A\rangle$$

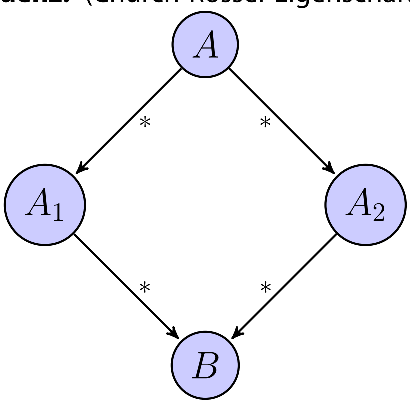
$$\Delta\langle W|S|\underline{output}\ T.K|E|A\rangle = \langle W|S|T.\underline{output}.K|E|A\rangle$$
$$\Delta\langle n.W|S|\underline{output}.K|E|A\rangle = \langle W|S|K|E|n.A\rangle$$

4 Reduktionssemantik (Vorlesung 4 am 15.05.)

4.1 Reduktionssemantik

Ausprägungen der <u>operationellen Semantik</u> zu einfacheren Argumentation (Beweisführung) über Programmeigenschaften.
Idee: Reduktion von Ausdrücken, Termen, Programmen und Anweisungen (usw.) auf einfachere aber semantisch äquivalenten Termen (usw.).
Beispiel: Einfache arithmetische Ausdrücke über den natürlichen Zahlen und +, *.
Einzelschrittreduktion: $(4+2)*(7-5) \Rightarrow 6*(7-5) \Rightarrow 6*2 \Rightarrow 12$
allgemeine Form: $A_1\ \underline{OP}\ A_2 \Rightarrow A_1'\ \underline{OP}\ A_2'$ falls $A_1 \Rightarrow A_1'$ und $A_2 \Rightarrow A_2'$
Axiom: $n_1\ OP\ n_2 \Rightarrow \underline{n_1\ OP\ n_2}$
Alternative Schreibweise(Winskel): $\frac{A_1 \Rightarrow A_1',\, A_2 \Rightarrow A_2'}{A_1\ \underline{OP}\ A_2 \Rightarrow A_1'\ \underline{OP}\ A_2'}$
Konfluenz: (Church-Rosser Eigenschaft, Diamant)

Church-Rosser gilt nicht bei Nebenwirkungen!



anderes Beispiel: λ-Kalkül. $(\lambda x.M)A \xrightarrow{\beta} \$M$ falls...

4.2 Reduktionssemantik der Sprache WHILE

Zustandsraum: $\mathcal{Z} = \rho \times \mathcal{E} \times \mathcal{A}$ Speicher, Ein- und Ausgabe. $\rho = [ID \to \text{ZAHL} \cup \{\underline{\text{frei}}\}], \mathcal{E} = \text{KON}^*, \mathcal{A} = \text{KON}^*$
$\Rightarrow$ Reduktionsrelation über (TERM $\cup$ BT $\cup$ COM) $\times \mathcal{Z}$
Induktiv über den Aufbau der Syntax:

1. **Terme**: Keine Reduktionsregel für (n, z), d.h. Normalform für $n \in$ ZAHL
   - **a** $(x, (s, e, a)) \Rightarrow (s(x), (s, e, a))$, falls $s(x) \neq \underline{\text{frei}}$ für $x \in ID$, $(s, e, a) \in \mathcal{Z}$
   - **b** <u>read</u> $\Rightarrow (n, (s, e, a))$, falls $n \in$ ZAHL.
   - **c** $(T_1\ \underline{OP}\ T_2, z) \Rightarrow (n\ \underline{OP}\ T_2, z')$, falls $(T_1, z) \overset{*}{\Rightarrow} (n, z')$
   - **d** $(n\ \underline{OP}\ T_1, z) \Rightarrow (n\ \underline{OP}\ m, z')$, falls $(T, z) \overset{*}{\Rightarrow} (m, z')$
   - **e** $(n\ \underline{OP}\ n, z) \Rightarrow (\underline{n\ OP\ m}, z')$, falls $\underline{n\ OP\ m} \in$ ZAHL.

2. **BT** analog.

3. **COM**: keine Reduktionsregel für $\underline{skip}$ (Normalform)
   - **a** $(I := T, (s, e, a)) \Rightarrow (\underline{skip}, (s[n/I], e', a))$, falls $(T, (s, e, a)) \overset{*}{\Rightarrow} (n, (s, e', a))$
   - **b** $\underline{output}\ T, (s, e, a) \Rightarrow (\underline{skip}, (s, e', a.n))$, falls $(T, (s, e, a)) \overset{*}{\Rightarrow} (n, (s, e', a))$
   - **c** $\underline{output}\ B, (s, e, a) \Rightarrow (\underline{skip}, (s, e', a.b))$, falls $(B, (s, e, a)) \overset{*}{\Rightarrow} (b, (s, e', a))$
   - **d** $(C_1; C_2, z) \Rightarrow (C_2, z)$, falls $(C_1, z) \overset{*}{\Rightarrow} (\underline{skip}, z')$
   - **e** $(\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2, z) \Rightarrow (C_1, z')$, falls $(B, z) \overset{*}{\Rightarrow} (\underline{true}, z')$
   - **f** $(\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2, z) \Rightarrow (C_2, z')$, falls $(B, z) \overset{*}{\Rightarrow} (\underline{false}, z')$
   - **g** $(\underline{while}\ B\ \underline{do}\ C, z) \Rightarrow (C; \underline{while}\ B\ \underline{do}\ C, z')$, falls $(B, z) \overset{*}{\Rightarrow} (\underline{true}, z')$

h $(\underline{while}\ B\ \underline{do}\ C, z) \Rightarrow (\underline{skip}, z')$, falls $(B, z) \overset{*}{\Rightarrow} (\underline{false}, z')$

4.3 Reduktionssemantik (schematisch)

$$\underline{eval}(P)(E) = \begin{cases} A, & \text{falls } (P, (S_0, E, \mathcal{E})) \overset{*}{\Rightarrow} (\underline{skip}, (S, E', A)) \text{ mit bel. } S \in \rho \text{ und } E', A \in \text{KON}^* \\ \underline{\text{Fehler}}, & \text{falls } (P, (S_0, E, \mathcal{E})) \overset{*}{\Rightarrow} (C, (S, E', A)) \text{ mit bel. } C \in \text{ COM und } E', A \in \text{KON}^* \\ & \text{und } C \neq \underline{skip} \text{ und } (C, (S, E', A)) \text{ lässt sich nicht mehr mit } \Rightarrow \text{ reduzieren} \\ \text{undefiniert}, & \text{sonst.} \end{cases}$$

Satz: $\mathcal{O} = \underline{eval}$ (extensional)

Beweis über strukturelle Induktion.

5 Denotationelle Semantik (Vorlesung 5 am 22.05.)

Grundprinzip:
Direkte (injektive) Zuordnung von syntaktischen Strukturen zu deren Semantik (als mathematisches Objekt (Konstrukt)).

5.1 Allgemeines Prinzip

1. Ordne jeder syntaktischen Kategorie K der Sprache einen semantischen Bereich B_K zu.
2. Schreibe (definiere) zu jeder syntaktischen Kategorie eine Semantikfunktion $\mathcal{K}$
   $\mathcal{K} : K \to B_K$

Notation:
- Das Argument einer Semantikfunktion $\mathcal{K}$ schreiben wir in $[\![$ und $]\!]$ *(Klammern sparen!)*
- Funktionen z.B. $f : A \to (B \to C)$ werden $f\ a\ b = a \dots b \dots$

5.2 Denotationelle Semantik

- **Syntaktische Kategorien** von WHILE: TERM, BT, COM, PROG
- Hilfsbereiche:
  ZUSTAND = SPEICHER $\times$ EINGABE $\times$ AUSGABE mit SPEICHER = ID $\to$ ZAHL $\cup\{\underline{frei}\}$,
  EINGABE = KON* und AUSGABE = KON* *(Abh. vom Speicher + Nebenwirkung)*

1. $\mathcal{T}$:TERM$\to$ $\underline{\underline{B_T}}$
   B_T :=ZAHL$\overline{???}$

Exkurs: **Curry-Isomorphie**

$$\begin{tikzcd}
(A \times B \to C) \arrow[rr, bend left, "\text{curry}"] & \cong & A \to (B \to C) \arrow[ll, bend left, "\text{uncurry}"]
\end{tikzcd}$$

curry: $f\ a\ b = f(a, b)$
uncurry: $g\ (a, b) = g\ a\ b$
B_T := SPEICHER $\to$ EINGABE $\to$ ((ZAHL $\times$ Eingabe) $\cup\{\underline{Fehler}\}$)
$\mathcal{B}$: BT $\to$ SPEICHER $\to$ EINGABE $\to$ ((BOOL $\times$ Eingabe) $\cup\{\underline{Fehler}\}$)
$\mathcal{C}$: COM $\to$ $\underbrace{\text{ZUSTAND} \to \text{ZUSTAND} \cup \{\underline{Fehler}\}\ )}_{B_C}$

2. Induktive Definition über dem Aufbau

$$\mathcal{T}[\![n]\!]\ s\ e = (n, e) \quad \text{für alle } n \in \text{ZAHL}$$

$$\mathcal{T}[\![x]\!]\ s\ e = \begin{cases} (n, e), & \text{falls } s\ x = n \\ \underline{\text{Fehler}}, & \text{falls } s\ x = \underline{frei} \text{ für alle } x \in \text{ID} \end{cases}$$

$$\mathcal{T}[\![\underline{read}]\!]\ s\ e = \begin{cases} (n, e'), & \text{falls } e = n.e' \\ \underline{\text{Fehler}}, & \text{falls } e = \mathcal{E} \text{ oder } e = b.e' \text{ mit } b \in \text{BOOL} \end{cases}$$

$$\mathcal{T}[\![T_1\ \underline{OP}\ T_2]\!]\ s\ e = \begin{cases} (n, e''), & \text{falls} \quad \mathcal{T}[\![T_1]\!]se = (n_1, e') \text{ und} \\ & \qquad\quad \mathcal{T}[\![T_2]\!]se' = (n_2, e'') \text{ und} \\ & \qquad\quad \underline{n_1\ \underline{OP}\ n_2} = n \in \text{ZAHL} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![w]\!]\ s\ e = (w, e) \quad \text{für alle} w \in \text{BOOL}$$

$$\mathcal{B}[\![\underline{not}B]\!]\ s\ e = \begin{cases} (b, e'), & \text{falls } \mathcal{B}[\![B]\!]se = (w, e') \text{ und } b = \neg w \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![T_1\ \underline{BOP}\ T_2]\!]\ s\ e = \begin{cases} (b, e''), & \text{falls} \quad \mathcal{T}[\![T_1]\!]se = (n_1, e') \text{ und} \\ & \qquad\quad \mathcal{T}[\![T_2]\!]se' = (n_2, e'') \text{ und} \\ & \qquad\quad \underline{n_1\ \underline{BOP}\ n_2} = b \in \text{BOOL} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![\underline{read}]\!]\ s\ e = \begin{cases} (b, e'), & \text{falls } e = b.e' \text{ mit } b \in \text{BOOL} \\ \underline{\text{Fehler}}, & \text{falls } e = \mathcal{E} \text{ oder } e = n.e' \text{ mit } n \in \text{ZAHL} \end{cases}$$

$$\mathcal{C}[\![\underline{skip}]\!]\ z = z$$

alt. Schreibweise: $\mathcal{C}[\![\underline{skip}]\!]\ z = \underline{id}$

$z = (s, e, a)$

$$\mathcal{C}[\![I := T]\!]\ (s, e, a) = \begin{cases} (S[n/I], e', a), & \text{falls } \mathcal{T}[\![T]\!]\ s\ e = (n, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{output}\ T]\!]\ (s, e, a) = \begin{cases} (s, e', a.n), & \text{falls } \mathcal{T}[\![T]\!]se = (n, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{output}\ B]\!]\ (s, e, a) = \begin{cases} (s, e', a.b), & \text{falls } \mathcal{B}[\![B]\!]se = (b, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![C_1; C_2]\!]\ z = \begin{cases} \mathcal{C}[\![C_2]\!]z', \mathcal{C}[\![C_1]\!]z = z' & \text{mit } z' \in \text{ZUSTAND} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{if}\ B\underline{then}\ C_1\ \underline{else}\ C_2]\!]\ (s, e, a) = \begin{cases} \mathcal{C}[\![C_1]\!]\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{true}, e') \\ \mathcal{C}[\![C_2]\!]\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{false}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{while}\ B\ \underline{do}\ C]\!]\ (s, e, a) = \begin{cases} \mathcal{C}[\![C; \underline{while}\ B\ \underline{do}\ C]\!](s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{true}, e') \\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{false}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{P}[\![P]\!]\ e = \begin{cases} a, & \text{falls} \quad \mathcal{C}[\![C]\!](s_0, e, \epsilon) = (s, e', a) \\ \text{Fehler}, & \text{falls} \quad \mathcal{C}[\![C]\!](s_0, e, \epsilon) = \underline{Fehler} \\ \text{undefiniert sonst} & \end{cases}$$

6 Axiomatische Semantik (Vorlesung 6 am 05.06.)

Am Beispiel von WHILE' ($\underline{read}\ I$ anstelle von Termen der Form $\underline{read}$).
Grundlage: C. A. R. Hoare: *An aximatic Basis for Computer Programming* (CACM 1969)

(Hoare-Kalkül zum Bew. von Prog. Eigenschft.)

6.1 Allgemeine Methode

Annahme: AST ist gegeben.
Neben der abstrakten Syntax brauchen wir vier Angaben:

1. Eine Menge von Bedingungen (*logische Ausdrücke, Prädikate, assertions*), die über den Zustandsraum interpretierbar sind. Im Allgemeinen Prädikatenlogik 1. Stufe oder Aussagenlogik.
   (formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)
   Beispiel: $x < y, input = \epsilon, \exists i.x = p.i, \ldots$ natürliche Interpretation: Verbinde $I \in$ ID mit $S(I)$ sowie $\underline{input}$ mit Eingabe....
   Sei $S(x) = 0$, dann ist die Bedingung $y/x = 7$ falsch.

   (formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)

2. Definition zu jeder atomaren Anweisung $\mathcal{C}$ ein Axiom bzw. ein Axiomenschema der Form $\{Q\}C\{R\}$ - Hoare Formel
   zu lesen als: Wenn die Bedingung Q auf einem Zustand z gilt und die Ausführung von C auf z in einem Zustand z' terminiert, dann gilt R auf z'.
   Beispiel: für $\underline{skip}$ des Axiomenschema: $\{Q\}\ \underline{skip}\ \{Q\}$, *zu lesen: für alle Q gilt es.*

3. Für zusammengesetzte Anweisungen C mit unmittelbaren Komponenten $C_1, \ldots, C_r$ eine Schlussregel der Form $\frac{F_1,\ldots,F_i}{\{Q\}C\{R\}}$, wobei die F_i Hoare-Formeln zu $C1, \ldots, C_r$ oder andere Formeln sind.

4. *Logischer Klebstoff*: Allgemeine Schlussregeln in Verbindung mit Hoare-Formeln, mindestens (oft ausreichend) die Konsequenzregel: $\frac{Q \Rightarrow S, T \Rightarrow R, \{S\}C\{T\}}{\{Q\}C\{R\}}$, wobei $\Rightarrow$ die logische Herleitung bezeichnet. Daraus folgt: Beweisbar $\Rightarrow$ Gültig. Frage: gilt diese Herleitung auch andersrum (F gültig $\Rightarrow$ beweisbar)? Nein. Im Allgemeinen landet man beim Haltemaschinenproblem und dem Gödelschen Unvollständigkeitssatz.

6.2 Konkretisierung am Beispiel von WHILE'

$$T ::= Z \mid I \mid T_1\ \underline{OP}\ T_2$$
$$B ::= \underline{true} \mid \underline{false} \mid \underline{eof} \mid T_1\ \underline{Relop}\ T_2$$
$$C ::= \underline{skip} \mid \underline{read}\ I \mid I := T \mid \underline{output}\ T \mid C_1; C_2 \mid \underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2 \mid \underline{while}\ B\ \underline{do}\ C$$

1. Aussagenlogik:
   - Terme über $Z, I, OP, \underline{input}, \underline{output}$, Konstruktor- und Selektorfunktionen $(., \underline{hd}, \underline{tl})$
   - Bedingungen: $T_1\ \underline{Relop}\ T_2$
     $B_1 \vee B_2, B_1 \wedge B_2, \neg B$

2. atomare Definitionen:

$$\{Q\}\underline{skip}\{Q\} \tag{A.1}$$

(zu lesen: Für jede Bedingung Q gilt die Hoare-Formel $\{Q\}\underline{skip}\{Q\}$)

$$\{Q[T/I]\}I := T\{Q\},$$

wobei$[T/I]$ die einmalige textliche Substitution von T für jedes Vorkommen von I in Q bezeichnet.

(A.2)

$$\{Q[\underline{hd}\ \underline{input}/I \mid \underline{tl}input/\underline{input}]\ \underline{read}\ I\{Q\} \approx I := \underline{hd}(\underline{input}); \underline{input} := \underline{tl}\ (\underline{input}) \tag{A.3}$$

$$\{Q[output.T/\underline{output}]\}\ \underline{output}\ T\{Q\} \approx \underline{output} = \underline{output}.T \tag{A.4}$$

Beispiele:

$$\{input = (4,5) \wedge 2 + x = 3\}I := 2 + x\{input = (4,5) \wedge I = 3\}$$

$$\{\underline{hd}\ \underline{input} = 7 \wedge \underline{tl}\ \underline{tl}\ \underline{input} = (3)\}\ \underline{read}\ x\{x = 7 \wedge input = (3)\}$$

$$\{\underline{output}.7 + y = (1,2,3,4,5)\}\ \underline{output}(7+9)\{\underline{output} = (1,2,3,4,5)\}$$

3.

$$\frac{\{Q\}C_1\{S\}, \{S\}C_2\{R\}}{\{Q\}C_1, C_2\{R\}} \tag{A.5}$$

$$\frac{\{Q \wedge B\}C_1\{R\}, \{Q \wedge \neg B\}C_2\{R\}}{\{Q\}\underline{if}B\underline{then}C_1\underline{else}C_2\{R\}} \tag{A.6}$$

$$\frac{\{Q \wedge B\}C\{Q\}}{\{Q\}\underline{while}\ B\ \underline{do}\ C\{Q \wedge \neg B\}} \tag{A.7}$$

4. Schlussregeln:

7 Mathematische Grundlagen zur Konstruktion sematischer Bereiche (Vorlesung 7 am 12.06.)

7.1 Den. Semantik

Den. Semantik: Zuordnung von mathematischen Objekten zu syntaktischen Konstrukten.
Problem: Rekursion. (Wdh.: Wir erklären eine Funktion (Zustand $\rightarrow$ Zustand $\cup$ Fehler)...

$$\mathcal{C}[\![\underline{while}\ B\ \underline{do}\ C]\!]\ (s,e,a) = \begin{cases} \mathcal{C}[\![C;\underline{while}\ B\ \underline{do}\ C]\!](s,e',a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{true}, e') \\ (s,e',a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{false}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

Frage: Gibt es eine eindeutige Lösung solcher Rekursionsgleichungen?
Antwort: I.A. nein.
Beispiele: (von leicht zu schwer)

$f(x) = f(x) + 1$ Keine Lösung im Bereich der totalen Funktionen $\mathrm{N} \rightarrow \mathrm{N}$.
Eine Lösung im Bereich der partiellen Fuktionen $\mathrm{N} \rightarrow \mathrm{N}$
$f(x)$ ist undefiniert für alle $x \in \mathrm{N}$.

(Beispiel 1)

$$f(x) = \begin{cases} 0, & \text{falls} f(x) = 0 \\ 1, & \text{falls} f(x) \neq 0 \end{cases}$$

Drei Lösungen über partiellen Funktionen $\mathrm{N} \rightarrow \mathrm{N}$:
a) $n \mapsto 0$
b) $n \mapsto 1$
c) $f(n)$ ist nicht definiert für alle $n \in \mathrm{N}$

(Beispiel 2)

$$f(x) = \begin{cases} y, & \text{falls} f(x) = 0 \\ f(f(x,y-1), f(x-1,y)) & \text{, sonst} f(x) \neq 0 \end{cases}$$

Lösungen im Breich der partiellen Funktionen $\mathrm{N} \times \mathrm{N} \rightarrow \mathrm{N}$:
a) $(x,y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ \text{undefiniert sonst} \end{cases}$
b) $(x,y) \rightarrow y$
c) $(x,y) \mapsto \max(x,y)$
d) $(x,y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ k, & \text{sonst für } k \in \mathrm{N}_+ \text{ bel.} \end{cases}$

(Beispiel 3)

Sei $g : \mathrm{N} \rightarrow \mathrm{N}$ eine idempotente Funktion mit$g(x) \neq 0$, d.h.
$g(gx) = g\ x$, dann ist

$$(x,y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ g(x), & \text{sonst} \end{cases}$$

(eine Lösung zu Beispiel 3)

7.2 Konsequenz

Konstruiere semantische Bereiche so, dass zu jeder rekursiven Gleichung eine *eindeutige Lösung* zugeordnet werden kann.
Idee:

- Betrachte totale Funktion über Bereichen, in denen ein Element undefiniert ($\perp$) hinzugefügt ist.
- Betrachte eine Approximationsrelation $\sqsubseteq$(weniger definiert als), um unendliche Objekte durch endliche zu approximieren.

Beispiel

$$\text{Part. Fkt. } \mathrm{N} \to \mathrm{N} \text{ mit } f \sqsubseteq g \text{ gdw } Graph(f) \leq Graph(g)$$
$$\text{doppelt } x = 2 * x \text{ wird approximiert von}$$
$$f_{\sqsubseteq}(x) \begin{cases} 2 * x, & \text{falls } x < \sqsubseteq \\ \text{undefiniert sonst}, \sqsubseteq \in \mathrm{N} \end{cases}$$

7.3 Defintion Semantischer Bereich, CPO

Eine Struktur $\underline{A} = (A, \sqsubseteq_A)$ heißt semantischer Bereich (cpo), wenn 1-3 gilt:

cpo - complete partial order

1. $\sqsubseteq_A$ ist eine Halbordnung auf A (reflexiv, transitiv, antisymmetrisch)
2. Es gibt bezüglich $\sqsubseteq_A$ ein minimales Element $\perp_A$ in A, d.h. $\perp_A \sqsubseteq a$, für alle $a \in A$
3. Zu jeder Kette $K \leq A$ existiert eine kleinste obere Schranke $\sqcup K$ in A, wobei K Kette ist, wenn zu je zwei $k_1, k_2 \in K$ gilt: $k_1 \sqsubseteq_A k_2$ oder $k_2 \sqsubseteq_A k_1$.

Beispiel

$(\mathrm{N}, \leq)$ ist cpo? Nein! 1. ✓ 2. $0 = \perp_{\mathrm{N}}$ ✓3. $\{x | x$ ist gerade$\}$ist Kette in N hat in N keine kleinste obere Schranke!$\times$

7.4 Defintionen (monton, stetig, strikt, [$\to$])

Seien A und B cpo's.

- Eine Funktion $f : A \to B$ heißt **monton**, wenn $f(a_1) \sqsubseteq_A f(a_2)$ für alle $a_1 \sqsubseteq a_2 \in A$
- Eine Funktion $f : A \to B$ heißt **stetig**, wenn $f(K)$ eine Kette in B ist und $f(\bigsqcup K) = \bigsqcup f(K)$ für alle $K \subseteq A$ mit K ist Kette in A.
- Eine Funktion $f : A \to B$ heißt **strikt**, wenn $f(\perp_A) = \perp_B$
- $[A \to B]$ bezeichnet den Raum aller stetigen Funktionen von $A \to B$.

1. Lemma $([A \to B], \sqsubseteq)$ ist cpo mit punktweiser Ordnung, d.h. $f \sqsubseteq g$, wenn $f(a) \sqsubseteq_B g(a)$ für alle $a \in A$

2. Lemma Aus f ist stetig folgt f ist monoton.

Beweis: Sei

$$f : A \to B \text{ stetig und } a_1 \sqsubseteq_A a_2 \in A$$
$$f(a_1) \sqsubseteq \sqcup\{f(a_1), f(a_2)\} = \sqcup f(\{a_1, a_2\}) = f(\sqcup\{a_1, a_2\}) = f(a_2) ✓$$

7.5 Satz: minimaler Fixpunkt (Tarski 1955)

Sei

$$f : A \to A \text{ eine stetige Funktion}$$
$$\text{Es existiert ein minimaler Fixpunkt}$$
$$\underline{fix}\ f \text{ in } A \text{ und es gilt}$$
$$\underline{fix}\ f = \bigsqcup_{\sqsubseteq \in \mathbb{N}} f^{\sqsubseteq}(\bot)$$

Nachtrag: Sei D ein cpo und $f : D \to D$ eine stetige Funktion. Es existiert in D der minimale Fixpunkt, $\underline{fix}f$, von f in D.
Es gilt $\underline{fix}\ f = \bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\bot)$.
Beweis:

$$\text{1.)} f(\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\bot)) = \bigsqcup_{\mu \in \mathbb{N}} f^{\mu+1}(\bot) \text{wg. Stetigkeit}$$
$$= \bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\bot)$$

Nebenbemerkung: $\{f^{\mu}(\bot) | \mu \in \mathbb{N}\}$ ist Kette in D.
Begründung: $\bot, f(\bot), f^2(\bot), \ldots, \bot \sqsubseteq f(\bot), f(\bot) \sqsubseteq f^2(\bot), f^2(\bot) \sqsubseteq f^3(\bot), \ldots$
Minimalität von $\bot$

Also ist $\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\bot)$ ein Fixpunkt von f.

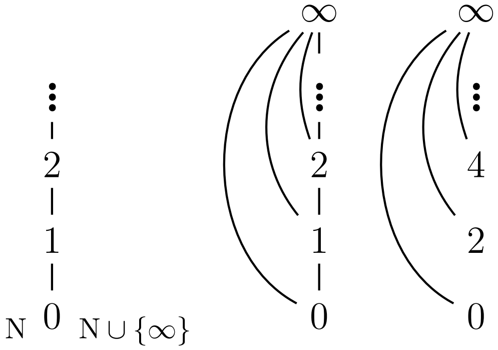
2.) (Minimalität) Sei $p \in D$ mit $f(p) = P$

$$\bot \sqsubseteq p \quad \text{Minimalität von } \bot$$
$$f(\bot) \sqsubseteq f(p) \quad \text{Lemma: Stetigkeit } \Rightarrow \text{ Monotonie}$$
$$f(\bot) \sqsubseteq f(p) \quad \text{für alle } \mu \in \mathbb{N}$$
$$\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\bot) \sqsubseteq p \quad \checkmark \quad \square$$

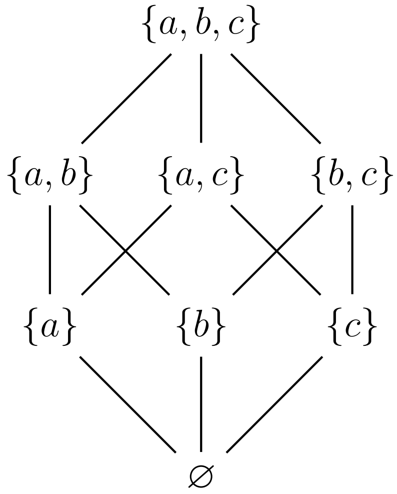
Ausblick: Rekursive Gleichungen können als Transformation von stetigen Funktionen betrachtet werden. *Fixpunkt dieser Transformation ist eindeutige Lösung.*

7.6 Graphische Illustration von cpo's

- Elemente des Trägers sind Knoten
- $\sqsubseteq$ wird durch aufwärts gerichtete Kanten dargestellt



Beispiel für cpo. Teilmengen von endlichen Mengen ($\mathcal{P}\{a,b,c\}, \subseteq$)



8 Konsturktion Semantischer Bereiche (cpo's) (Vorlesung 8 am 19.06.)

8.1 Abzählbare Mengen

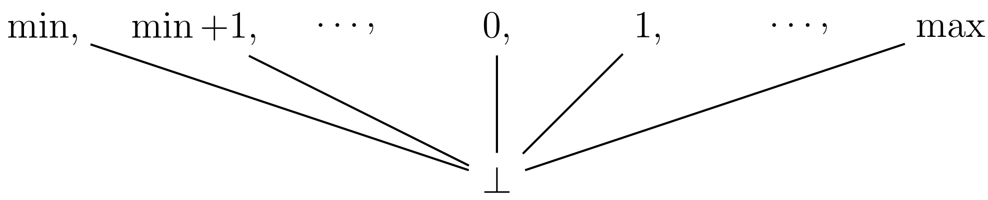
Sei M eine abzählbare Menge. Der cpo $M_\perp$ entsteht aus M durch

$$M_\perp = (M \cup \{\perp\}, \sqsubseteq) \text{ mit } x \sqsubseteq y \text{ gdw. } x = \perp \text{ oder } x = y$$

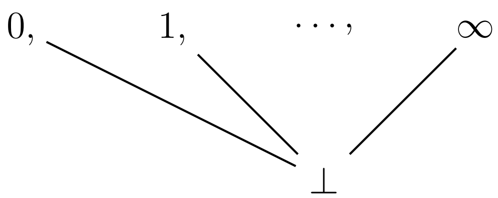
$\square$

Beispiele:

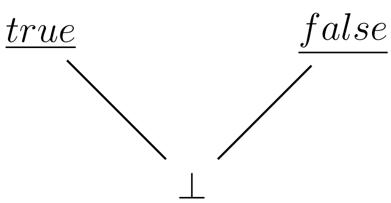
$$\text{ZAHL} = \{\min, ..., \max\}_\perp$$



$$\mathbb{N}_\perp = 0, 1, 2, ...$$



$$\text{BOOL} = \{\underline{true}, \underline{false}\}_\perp$$



$$\text{ID} = \{w | w \in \{a, b, \ldots, z\}^*\}_\perp$$

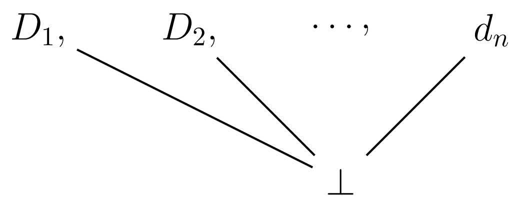
8.2 Kartesische Produkt

Seien $D_1, ..., D_n$ cpo's. $D = D_1 \times ... \times D_n$
D ist cpo mit komponentenweise Ordnung $\sqsubseteq_D$, d.h.

$$< d_1, ..., d_n > \sqsubseteq_D < d_1', ..., d_n' > \text{ gdw. } d_1 \sqsubseteq_{D_1} d_i' \text{ für alle } 1 \leq i \leq n$$
$$D := (D_1 \times D_n, \sqsubseteq_D) \text{ ist cpo.}$$

8.3 Summen

Seien: $D_1, ..., D_n$ cpo's



$D = D_1 + ... + D_n$ ist wie folgt erklärt.
$D = (\{(d,i) | d \in D_i, i \leq i \leq n\}, \sqsubseteq_D) \cup \{\perp_D\}, \sqsubseteq_D) mit (d,i) \sqsubseteq_D (e,j)$ gdw. $i = j$ und $\perp_D \sqsubseteq (d,i)$ für alle ...

8.4 endliche Folgen

Sei $\underline{D} = (D, \sqsubseteq_D)$ ein cpo.

$\underline{D^*} = (\{< d_i | d_i \in 1 \leq i \leq n > | n \in \mathbb{N}\} \cup \{\perp_{D^*}\}, \sqsubseteq_{D^*})$ mit

<u>Vorüberlegung</u>

$< d_i | 1 \leq i \leq n > \sqsubseteq_D < d_i' | 1 \leq i \leq m >$
gdw. $n \leq m$ und $d_i \sqsubseteq_D d_i'$ für alle $1 \leq i \leq n$
funktioniert nicht, weil
Sei $1 \in D$,
$K := \{<>, < 1 >, < 1, 1 >, < 1, 1, 1 >, ...\}$ Kette in D^*
$\bigsqcup K = < 1, 1, 1, ... >$ unendliche Folgen in D^W

Ende der Vorüberlegung

$< d_i | 1 \leq i \leq n > \sqsubseteq_{D^*} < d_i' | 1 \leq i \leq n >$
gdw. $n = m$ und $d_i \sqsubseteq_D d_i'$ für alle $1 \leq i \leq n$
$\perp_{D^*} \sqsubseteq_{D^*}$ für jede Folge $\in D^*$

d.h. $D^* = \bigcup_{i \in \mathbb{N}} D^i$

8.5 unendliche Folgen

$D^w := (\{< d_i | i \in \mathbb{N} >\}, \sqsubseteq_{D^W})$ mit $< d_i | i \in \mathbb{N} > \sqsubseteq_{D^w} < d_i' | i \in \mathbb{N} >$ gdw. $d_i \sqsubseteq_D d_i'$ für alle $i \in \mathbb{N}$

$$\perp_{D^W} = < \perp_D | i \in \mathbb{N} >$$

8.6 Funktionenraum

Seien D_1 und D_2 cpo's

$D = [D_1 \to D_2] := (\{f : D_1 \to D_2 | f$ ist stetig$\}, \sqsubseteq_D)$ mit punktweiser Ordnung, d.h.
$f \sqsubseteq_D g$ gdw. $f(d) \sqsubseteq_{D_2} g(d)$ für alle $d \in D_1$.

D ist cpo!

8.7 Ausblick

Konstruktionen rekrusiver Domains ist möglich. Wenn D eine cpo-Variable, dann löst sich eine Gleichung der Form $D = \tau[D, B_1, ..., B_r]$ lösen. τ über $\times, +, *, ^w, \to$, cpo's B_i
Kanonische Operationen, wie Projektionen, Injektionen, Test auf leere Liste, ..., sind stetig.

9 Der getypte λ-Kalkül als Metasprache (Vorlesung 9 am 26.06.)

- Semantische Bereiche, cpo's
- Bezeichnung für Elemente aus Ausgangsbereichen, kartesische Produkte, Folgen, Summenbereichen ist die vertraute mathematische Notation. Für Elemente aus Funktionenbereichen (-domains) bietet sich der λ -Kalkül von Alonso Church (1936) an.
  Bisher: $f : D_1 \to D_2$ wurde definiert durch $d \to t$, wobei in t sowohl der formale Parameter d als auch bekannte Elemente vorkommen können.
  Problematisch für Funktionen höheren Typs.

9.1 Definition: Menge A_λ der getypten λ -Ausdrücke

Sei $\mathcal{X} = \{\mathcal{X}^D | D \in \mathcal{D}, \mathcal{D}\}$ Familie von cpo's, die Ausgangsbereiche enthält und abgeschlossen ist unter: $x,^*, ^w, +und \to$.

typfrei: Sei $\mathcal{X}$ als unendl. Menge
(i) $x \in \mathcal{X}$ ist λ-Ausdruck (Atom)
(ii) ($t_1\ t_2$) ist λ-Ausdruck (Applikation)
(iii) ($\lambda x.t$) ist λ-Ausdruck (Abstraktion)

1.

$x : D \in \mathcal{A}_\lambda$ für alle $x \in \mathcal{X}^D$ (Atome, Variable)

$k : D \in \mathcal{A}_\lambda$ für alle $k \in \mathcal{K}^D$, wobei

$\mathcal{K} = \{\mathcal{K}^D | D \in D\}$, die diskrete Elemente, sowie Projektionsfunktion, Listenoperationen, $\underline{fix}, \underline{curry}, \ldots$

2.

$$< t_1, \ldots, t_r >: D_1 \times \cdots \times D_r \in \mathcal{A}_\lambda \text{ für alle } t_i : D_i \in \mathcal{A}_\lambda, 1 \le i \le r \qquad \text{(Tupel)}$$

3.

$$(t_1 t_2) : D \in \mathcal{A}_\lambda, \text{ falls } t_1 : D' \to D \in \mathcal{A}_\lambda und t_2 : D' \in \mathcal{A}_\lambda \qquad \text{(Applilation)}$$

4.a)

$$(\lambda x.t) : D_1 \to D_2 \in \mathcal{A}_\lambda, \text{ für alle } x \in \mathcal{X}^{D_1} undt : D_2 \in \mathcal{A}_\lambda \qquad \text{(monadisch)}$$

4.b)

$$(\lambda(x_1, \ldots, x_r).t) : D_1 \times \cdots \times D_i \to D, \text{ für } x_i \in \mathcal{X}^{D_i}, t : D \in \mathcal{A}_\lambda, 1 \le i \le r \qquad \text{(polyadisch)}$$

9.2 Konventionen (Verbesserung der Lesbarkeit)

- Applikation ist links assoziativ, d.h. $f\ a_1\ a_2$ steht für $(\ldots((f\ a_1)a_2)\ldots a_n)$
- Abstraktion erstreckt sich soweit nach rechts wie möglich, d.h. $\lambda x.t_1\ t_2 \ldots t_r$ steht für $(\lambda x.t_1\ t_2 \ldots t_r)$ und $t_0(\lambda x.t_1\ t_2 \ldots t_r)t_{r+1}$ steht für $(t_0(\lambda x.t_1\ t_2 \ldots t_r))t_{r+1}$
- Applikation mit rechtsassoziativer Bindung notiert durch ";" d.h. $t_1 \ldots t_r; s_1 \ldots s_n; u_1 \ldots u_m$ steht für $(t_1 \ldots t_r)((s_1 \ldots s_n)(u_1 \ldots u_m))$
- Mehrfachabstraktionen der Form $\lambda x_1.\lambda x_2 \ldots \lambda x_r.t$ wird abgekürzt durch $\lambda x_1 \ldots x_r.t$
- Die bekannten zweistelligen arithm. u. log. Operatoren werden Infix notiert und binden schwächer als die Applikation, d.h. $\lambda x.2 + fx$ steht für $(\lambda x.((\underline{plus}2)(fx)))$
- Verzicht auf Typenangabe, wenn aus Kontext ersichtlich!

Beispiel

Für die Verwendung getypter λ-Ausdruck zur Definition von Elementen aus funktionalen cpo's.
Verbal Gewicht f, welches angewendet auf eine Liste von Zahlen, eine zweistellige arithm. Operation g und eine Zahl z den Wert von g angewendet auf die dritte Komponente von L und z liefert.

$$f := \lambda L\ g\ z.g(\pi_3 L)z \text{ mit } g : \mathbb{Z}_p erp \to \mathbb{Z}_p erp \to \mathbb{Z}_p erp, L : D^* \text{oder} L : \mathbb{Z}_p^* erp, z : \mathbb{Z}_p erp$$
$$f : \mathbb{Z}_p erp^* \to [\mathbb{Z}_p erp \to \mathbb{Z}_p erp \to \mathbb{Z}_p erp] \to \mathbb{Z}_p erp \to \mathbb{Z}_p erp$$

gegeben folgenden Argumente $f < 4, 7, \underline{hd} < 3 >, 8, 2 > \underline{plus} 5 : \mathbb{Z}_p erp$
Rechnen durch Ersetzung von formalen Parametern durch akt. Argumente

$$\begin{aligned} f < 4, 7, \underline{hd} < 3 >, 8, 2 > \underline{plus} 5 &\to \underline{plus}(\pi_3 < \cdots >)5 \\ &\to \underline{plus}(\underline{hd} < 3 >)5 \\ &\to \underline{plus} 35 \\ &\to 8 \end{aligned}$$

9.3 Formale Semantik der Metasprache:

$$\begin{aligned} [\![\ ]\!] &: \mathcal{A}_\lambda \to [\mathcal{U} \to \mathcal{D}], \text{ wobei } \mathcal{U} : \mathcal{X} \to \mathcal{D} \text{ typhaltende Funktion.} \\ [\![x]\!]\rho &= \rho x \\ [\![k]\!]\rho &= k \end{aligned} \tag{1}$$

$$[\![< t_1, \ldots, t_n >]\!]\rho = < [\![t_1]\!]\rho, \ldots, [\![t_n]\!]\rho > \tag{2}$$

$$[\![t_1\ t_2]\!]\rho = [\![2_1]\!]\rho([\![t_2]\!]\rho) \tag{3}$$

$$[\![\lambda x.t]\!]\rho = d \to [\![t]\!]s[d/x] \tag{4a}$$

$$\begin{aligned} [\![\lambda(x_1, \ldots, x_r).t]\!]\rho &= < d_1, \ldots, d_r > \\ &\to [\![t]\!]\rho[d_1 \ldots d_r / x_1 \ldots x_i], x_i \neq x_j \text{ für alle } i \neq j \end{aligned} \tag{4b}$$

9.4 λ-Kalkül (Semantische Reduktionsregeln)

$Fr : \mathcal{A}_\lambda \to \mathcal{X}, Geb. : \mathcal{A}_\lambda \to \mathcal{X}, Var : \mathcal{A}_\lambda \to \mathcal{X}$

$\xrightarrow{\alpha}$ Variablenumbenennung $\lambda x.t \xrightarrow{\alpha} \lambda y.\$_y^x t$, wobei \$ Substitutionsoperator

$\xrightarrow{\beta}$ Ersetzung formaler Parameter $(\lambda x.t)a \xrightarrow{\beta} \$_a^x t$, falls $Fr(a)Geb(t) = \varnothing$

$\xrightarrow{\gamma}$ Wertreduktion, $\underline{plus} 4\ 3 \xrightarrow{\gamma} 7$

Extensionalität: $\lambda x.(t\ x) \xrightarrow{\eta} t, x \notin Fr(t)$

10 Die Metasprache λ-Kalkül (Vorlesung 10 am 03.07.)

10.1 Syntaktischer Zucker in der Metasprache

10.1.1 Kombinatoren (Namen für Ausdrücke)

$$\underline{id} = \lambda x.x \quad \text{Familie von Identitätsfunktion des Typs } D \to D \text{ für alle } D \in D$$
$$\underline{curry} : [(D_1 \times D_2) \to D_3] \to [D_1 \to D_2 \to D_3]$$
$$\underline{curry} = \lambda f\, x\, y.\, f < x, y > \quad \text{Schreibe } \underline{curry}\, f \text{ zur Transformation von } f$$
$$\vdots$$

Randbemerkung:
$S = \lambda xyz(xz)yz$
$K = \lambda xy.x$
$I = \lambda x.x$
und Applikation bilden berechenbare Funktionen

<u>Konvention</u>: Schreibe äußere Abstraktionen nach links, z.B.:

$$\underline{id}\, x = x$$
$$\underline{curry}\, f\, x\, y = f < x, y >$$

10.1.2 Operationen

Im Falle der 2-Stelligkeit auch in Infix-Notation.

$$\circ : ([D_2 \to D_3] \times [D_1 \to D_2]) \to [D_1 \to D_3]$$
$$(f \circ g)\, x = f(g\, x) \quad \text{alternativ: } \circ = \lambda(f,g)x.f(g,x)$$

$\bowtie$-Operator (gelesen "vor"), der Fehler weiterreicht und implizit $\underline{curry}$ anwendet.

<u>1. Fall</u> $f : D_1 \to (D_2 + \{\underline{Fehler}\}_\perp)$ und $g : D_2 \to (D_3 + \{\underline{Fehler}\}_\perp)$ oder $g : D_2 \to D_3$

<u>2. Fall</u> $f : D_1 \to (D_2 \times \cdots \times D_n) + \{\underline{Fehler}\}_\perp$ und $g : D_2 \to \cdots \to [D_n \to (D_{n+1} + \{\underline{Fehler}\}_\perp)]$ oder $g : D_2 \to \cdots \to D_n \to D_{n+1}$

zu 1.: $(f \bowtie g)x = f\ x = \underline{Fehler}, \underline{Fehler}$

$$f\ x = d, g\ h$$
$$\bowtie : [D_1 \to (D_2 + \{\underline{Fehler}\}_\perp)] \times [D_2 \to (D_3 + \{\underline{Fehler}\}_\perp)] \to [D_1 \to D_3 + \{\underline{Fehler}\}_\perp]$$

analog für $g : D_2 \to D_3$

zu 2.: $(f \bowtie g)\ x = f\ x = \underline{Fehler}, \underline{Fehler};$

$$f\ x = < d_2, \ldots, d_n > \to g\ d_2 \ldots d_n$$

10.1.3 Bedingte Ausdrücke

$\underline{cond}$ ist nicht strikt!

$$\underline{cond} : (D \times D) \to \text{BOOL} \to D$$
$$\underline{cond} < d_1, d_2 > b = \begin{cases} b = \underline{true} & \to d_1, \\ b = \underline{false} & \to d_2, \perp \end{cases}$$

Integration im getypten λ-Kalkül

Verwende $D \to D \to D$ anstelle von BOOL
kodiere $\underline{true} = \lambda xy.x$ $\underline{false} = \lambda xy.y$ $\underline{cond} < d_1, d_2 > b = b\, d_1\, d_2$

Musteranpassung in bedingten Ausdrücken

Wenn f aus einem Summebreich mit strukturell verschwiedenen Gleidern besteht, werden in- und out- und Termfunktionen implizit angewendet, z.B.

$$\begin{aligned} f : D_1 + (D_2 \times D_3) &\to D \\ f\ x = & x = d \to \ldots d \ldots, \\ & x =< d_1, d_2 >\to \ldots d_1 \ldots d_2 \ldots \end{aligned}$$

Wenn f nicht substituiert wird auch:

$$\begin{aligned} f\ d &= \ldots d \\ f < d_1, d_2 > &= \ldots d_1 \ldots d_2 \ldots \end{aligned}$$

10.1.4 Rekursionen

$f = A$ mit $f \in \mathcal{X}^D, A : D \in \mathcal{A}_\lambda$ definiert eindeutig ein Element in D.

wenn f in A frei vorkommt ist $f = A$ eine rekursive Gleichung, deren eindeutige Lösung durch $\underline{fix}\ \tau$, wobei $\tau = \lambda f.A$ gegeben ist!
Ordne $f = A$ die Transformation $\tau = \lambda f.A$ zu. $\tau : D \to D$.

$$\begin{aligned} \underline{fac}\ n &= n = 0 \to 1, n * \underline{fac}(n-1); \underline{fac} : \mathbb{N}_\perp \to \mathbb{N}_\perp \\ \tau &= \lambda fn.n = 0 \to 1, n * f(n-1) \\ \tau &: [\mathbb{N}_\perp \to \mathbb{N}_\perp] \to [\mathbb{N}_\perp \to \mathbb{N}_\perp] \text{ mit } \underline{fix}\ \tau : \mathbb{N}_\perp \to \mathbb{N}_\perp \\ \underline{\text{Behauptung:}} \quad \underline{fix}\ \tau &= ! \\ (\underline{fix}\ \tau) &= \bigsqcup_{\mu \in \mathbb{N}} \tau^\mu(\perp) \\ \tau^0(\perp) &= \lambda n.\ \perp \\ \tau^1(\perp)n &= n = 0 \to 1, \perp \\ \tau^2(\perp)n &= n = 0 \to 1, n = 1 \to 1, \perp \\ \tau^\mu(\perp)n &= n = 0 \to 1, n = 1 \to 1, \ldots, n = \mu - 1 \to (\mu - 1) \perp \\ \tau^{\mu+1}(\perp)n &= ... \end{aligned}$$

10.2 Syntaktische Bereiche von WHILE flache cpo's

10.2.1 Sematische Bereiche

$$\begin{aligned} \mathtt{SPEICHER} &= \mathtt{ID} \to (\mathtt{ZAHL} + \{\underline{frei}\}_\perp) \\ \mathtt{EINGABE} &= \mathtt{KON}^*, \mathtt{AUSGABE} = \mathtt{KON}^* \end{aligned}$$

10.2.2 Semantikfunktionen

$$
\begin{aligned}
\mathcal{T} &: \texttt{TERM} \to \texttt{ZUSTAND} \to ((\texttt{ZAHL} \times \texttt{ZUSTAND}) + \{\underline{Fehler}\}_\perp) \\
\mathcal{B} &: \texttt{TERM} \to \texttt{ZUSTAND} \to ((\texttt{BOOL} \times \texttt{ZUSTAND}) + \{\underline{Fehler}\}_\perp) \\
\mathcal{C} &: \texttt{COM} \to \texttt{ZUSTAND} \to (\texttt{ZUSTAND} + \{\underline{Fehler}\}_\perp) \\
\mathcal{P} &: \texttt{PROG} \to \texttt{EINGABE} \to (\texttt{EINGABE} + \{\underline{Fehler}\}_\perp) \\
\mathcal{T}[\![n]\!]z &= < n, z > && \text{für alle} n \in \texttt{ZAHL} \\
\mathcal{T}[\![x]\!] < s, e, a > &= s\,x = \underline{frei} \to \underline{Fehler}, < s\,x, < s, e, a >> \\
\mathcal{T}[\![\underline{read}]\!] < s, e, a > &= \underline{null}\,e \to \underline{Fehler}, < \underline{hd}\,e, < s, \underline{tl}\,e, a >> \\
\mathcal{T}[\![T_1 + T_2]\!] &= \mathcal{T}[\![T_1]\!] \bowtie \lambda n_1.\mathcal{T}[\![T_2]\!] \bowtie \lambda n_2 z. < n_1 + n_2, z > \\
\mathcal{C}[\![C_1, C_2]\!] &= \mathcal{C}[\![C_1]\!] \bowtie \mathcal{C}[\![C_2]\!]
\end{aligned}
$$

11 SOME BAD ASS STUFF TODAY (Vorlesung 11 am 10.07.)